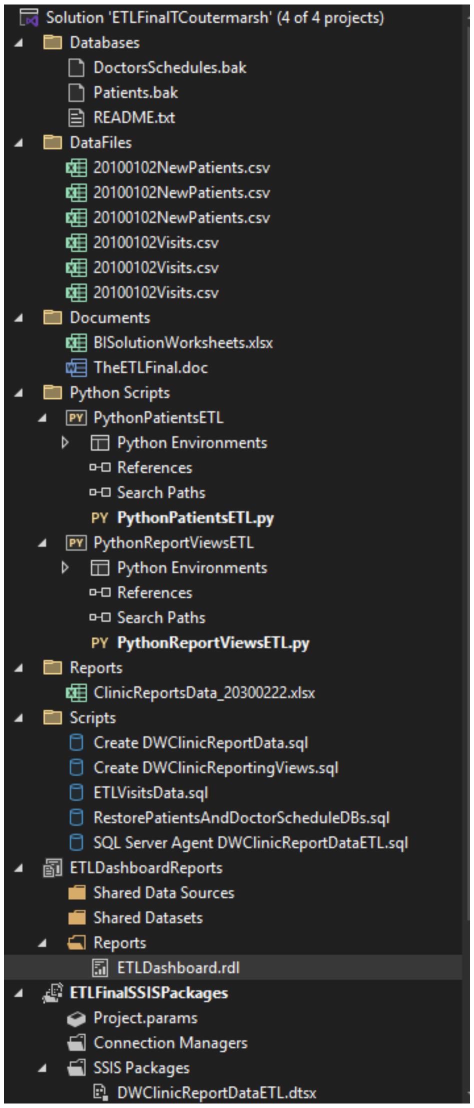
The ETL Final Admin Manual

Certificate in Business Intelligence and Database Development

Description: This manual outlines the components of the ETL Final business intelligence solution.
Change Log: (When, Who, What)
2025-03-27, TCoutermarsh, Created first draft of document

Overview

This BI solution is for a healthcare business that currently has a manual data uploading process for their three clinics**.** The solution consists of the following components:



```
Solution 'ETLFinalTCoutermarsh' (4 of 4 projects)
  Databases
    DoctorsSchedules.bak
    Patients.bak
    README.txt
  DataFiles
    20100102NewPatients.csv
    20100102NewPatients.csv
    20100102NewPatients.csv
    20100102Visits.csv
    20100102Visits.csv
    20100102Visits.csv
  Documents
    BISolutionWorksheets.xlsx
    TheETLFinal.doc
  Python Scripts
    PythonPatientsETL
      Python Environments
      References
      Search Paths
      PythonPatientsETL.py
    PythonReportViewsETL
      Python Environments
      References
      Search Paths
      PythonReportViewsETL.py
  Reports
    ClinicReportsData_20300222.xlsx
  Scripts
    Create DWClinicReportData.sql
    Create DWClinicReportingViews.sql
    ETLVisitsData.sql
    RestorePatientsAndDoctorScheduleDBs.sql
    SQL Server Agent DWClinicReportDataETL.sql
  ETLDashboardReports
    Shared Data Sources
    Shared Datasets
    Reports
      ETLDashboard.rdl
  ETLFinalSSISPackages
    Project.params
    Connection Managers
    SSIS Packages
      DWClinicReportDataETL.dtsx
```

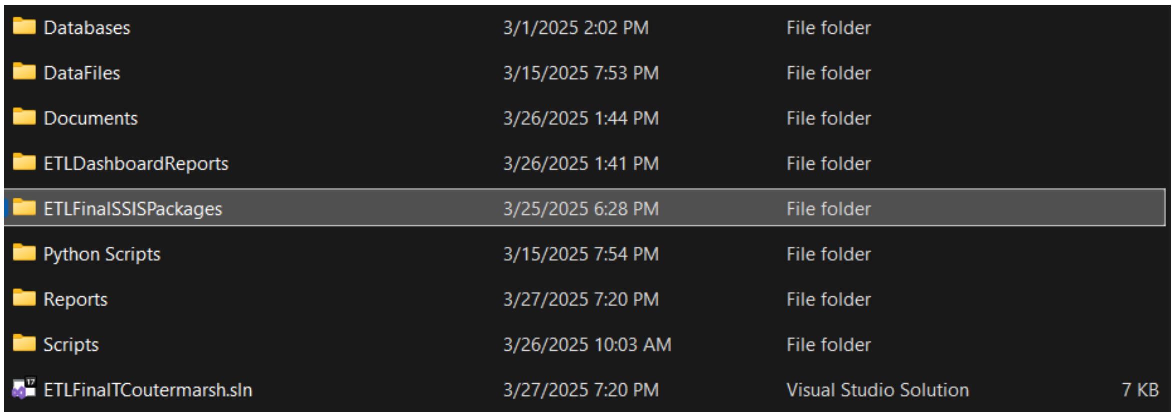
- The Visual Studio solution
- The BI Solution Worksheet
- The Business's Daily Data
- This Admin Manual Document
- The File-Based ETL
- The Data Warehouse ETL
- The Non-SQL ETL
- The SQL Server Agent Jobs
- The ETL Reporting Dashboard
- The Database Restoration Script

Figure: The components of the ETL Final BI Solution

The Visual Studio Solution

This component is used to store and organize all the other components of this BI Solution: SQL Scripts, Database backup files, Reports, Dashboards, Python Scripts, documents and SSIS packages. Visual Studio can connect to SQL Server and run the SQL Scripts if needed within the application if chosen. The python scripts within the BI Solution are also stored within Visual Studio and can be executed either from the script directly or from the SSIS packages. Within the physical folder of this BI Solution, Visual Studio mirrors all components and subfolders organization, and the Visual Studio solution is within the zipped physical folder.

Figure:

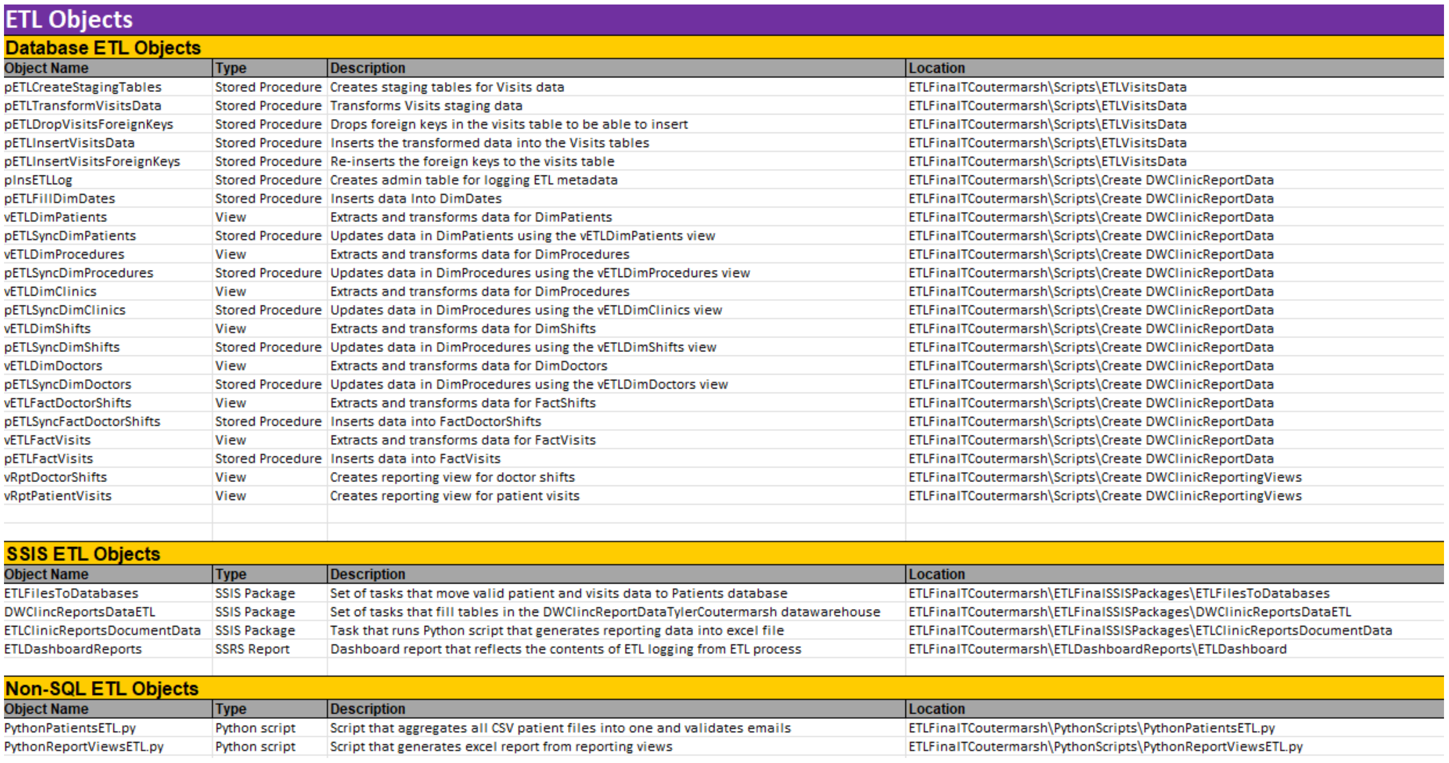


Databases	3/1/2025 2:02 PM	File folder	
DataFiles	3/15/2025 7:53 PM	File folder	
Documents	3/26/2025 1:44 PM	File folder	
ETLDashboardReports	3/26/2025 1:41 PM	File folder	
ETLFinalSSISPackages	3/25/2025 6:28 PM	File folder	
Python Scripts	3/15/2025 7:54 PM	File folder	
Reports	3/27/2025 7:20 PM	File folder	
Scripts	3/26/2025 10:03 AM	File folder	
ETLFinalTCoutermarsh.sln	3/27/2025 7:20 PM	Visual Studio Solution	7 KB

The BI Solution Worksheet

This component is used to document other components of the ETL process. In the first tab called “ETL Transformations”, this tracks transformations from the patients and visits data of the three clinics to the database. As well as the transformations from the database to the DWClinicReportData Datawarehouse. The tab, “ETL Objects” documents the Database ETL Objects, SSIS ETL Objects and the Non-SQL ETL Objects.

Figure:



ETL Objects

Database ETL Objects

Object Name	Type	Description	Location
pETLCreateStagingTables	Stored Procedure	Creates staging tables for Visits data	ETLFinalTCoutermarsh\Scripts\ETLVisitsData
pETLTransformVisitsData	Stored Procedure	Transforms Visits staging data	ETLFinalTCoutermarsh\Scripts\ETLVisitsData
pETLDropVisitsForeignKeys	Stored Procedure	Drops foreign keys in the visits table to be able to insert	ETLFinalTCoutermarsh\Scripts\ETLVisitsData
pETLInsertVisitsData	Stored Procedure	Inserts the transformed data into the Visits tables	ETLFinalTCoutermarsh\Scripts\ETLVisitsData
pETLInsertVisitsForeignKeys	Stored Procedure	Re-inserts the foreign keys to the visits table	ETLFinalTCoutermarsh\Scripts\ETLVisitsData
pInsETLLog	Stored Procedure	Creates admin table for logging ETL metadata	ETLFinalTCoutermarsh\Scripts\Create DWClinicReportData
pETLFillDimDates	Stored Procedure	Inserts data Into DimDates	ETLFinalTCoutermarsh\Scripts\Create DWClinicReportData
vETLDimPatients	View	Extracts and transforms data for DimPatients	ETLFinalTCoutermarsh\Scripts\Create DWClinicReportData
pETLSyncDimPatients	Stored Procedure	Updates data in DimPatients using the vETLDimPatients view	ETLFinalTCoutermarsh\Scripts\Create DWClinicReportData
vETLDimProcedures	View	Extracts and transforms data for DimProcedures	ETLFinalTCoutermarsh\Scripts\Create DWClinicReportData
pETLSyncDimProcedures	Stored Procedure	Updates data in DimProcedures using the vETLDimProcedures view	ETLFinalTCoutermarsh\Scripts\Create DWClinicReportData
vETLDimClinics	View	Extracts and transforms data for DimProcedures	ETLFinalTCoutermarsh\Scripts\Create DWClinicReportData
pETLSyncDimClinics	Stored Procedure	Updates data in DimProcedures using the vETLDimClinics view	ETLFinalTCoutermarsh\Scripts\Create DWClinicReportData
vETLDimShifts	View	Extracts and transforms data for DimShifts	ETLFinalTCoutermarsh\Scripts\Create DWClinicReportData
pETLSyncDimShifts	Stored Procedure	Updates data in DimProcedures using the vETLDimShifts view	ETLFinalTCoutermarsh\Scripts\Create DWClinicReportData
vETLDimDoctors	View	Extracts and transforms data for DimDoctors	ETLFinalTCoutermarsh\Scripts\Create DWClinicReportData
pETLSyncDimDoctors	Stored Procedure	Updates data in DimProcedures using the vETLDimDoctors view	ETLFinalTCoutermarsh\Scripts\Create DWClinicReportData
vETLFactDoctorShifts	View	Extracts and transforms data for FactShifts	ETLFinalTCoutermarsh\Scripts\Create DWClinicReportData
pETLSyncFactDoctorShifts	Stored Procedure	Inserts data into FactDoctorShifts	ETLFinalTCoutermarsh\Scripts\Create DWClinicReportData
vETLFactVisits	View	Extracts and transforms data for FactVisits	ETLFinalTCoutermarsh\Scripts\Create DWClinicReportData
pETLFactVisits	Stored Procedure	Inserts data into FactVisits	ETLFinalTCoutermarsh\Scripts\Create DWClinicReportData
vRptDoctorShifts	View	Creates reporting view for doctor shifts	ETLFinalTCoutermarsh\Scripts\Create DWClinicReportingViews
vRptPatientVisits	View	Creates reporting view for patient visits	ETLFinalTCoutermarsh\Scripts\Create DWClinicReportingViews

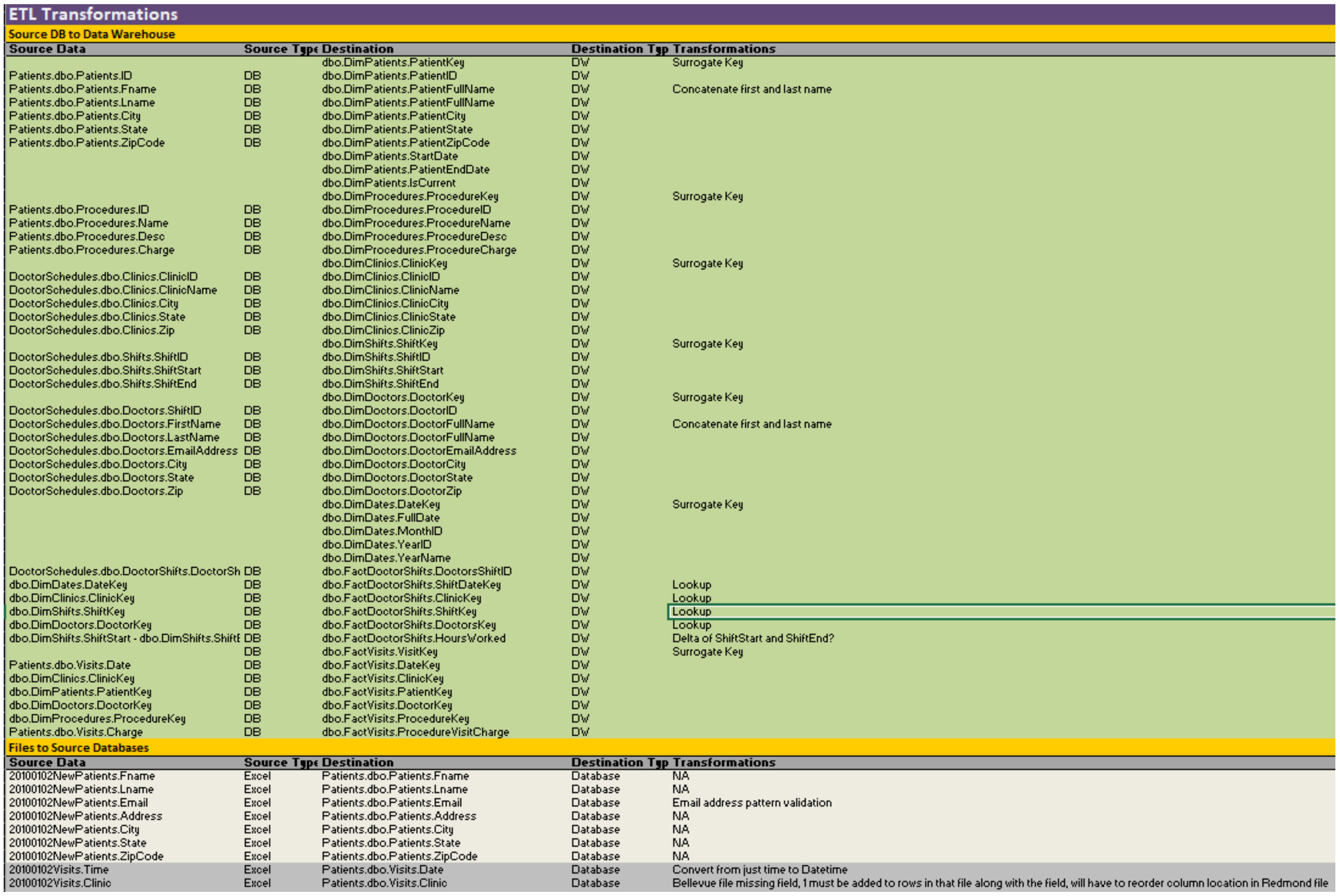
SSIS ETL Objects

Object Name	Type	Description	Location
ETLFilesToDatabases	SSIS Package	Set of tasks that move valid patient and visits data to Patients database	ETLFinalTCoutermarsh\ETLFinalSSISPackages\ETLFilesToDatabases
DWClincReportsDataETL	SSIS Package	Set of tasks that fill tables in the DWClincReportDataTylerCoutermarsh datawarehouse	ETLFinalTCoutermarsh\ETLFinalSSISPackages\DWClinicReportsDataETL
ETLClinicReportsDocumentData	SSIS Package	Task that runs Python script that generates reporting data into excel file	ETLFinalTCoutermarsh\ETLFinalSSISPackages\ETLClinicReportsDocumentData
ETLDashboardReports	SSRS Report	Dashboard report that reflects the contents of ETL logging from ETL process	ETLFinalTCoutermarsh\ETLDashboardReports\ETLDashboard

Non-SQL ETL Objects

Object Name	Type	Description	Location
PythonPatientsETL.py	Python script	Script that aggregates all CSV patient files into one and validates emails	ETLFinalTCoutermarsh\PythonScripts\PythonPatientsETL.py
PythonReportViewsETL.py	Python script	Script that generates excel report from reporting views	ETLFinalTCoutermarsh\PythonScripts\PythonReportViewsETL.py

Figure:



ETL Transformations

Source DB to Data Warehouse

Source Data	Source Type	Destination	Destination Typ	Transformations
		dbo.DimPatients.PatientKey	DW	Surrogate Key
Patients.dbo.Patients.ID	DB	dbo.DimPatients.PatientID	DW	
Patients.dbo.Patients.Fname	DB	dbo.DimPatients.PatientFullName	DW	Concatenate first and last name
Patients.dbo.Patients.Lname	DB	dbo.DimPatients.PatientFullName	DW	
Patients.dbo.Patients.City	DB	dbo.DimPatients.PatientCity	DW	
Patients.dbo.Patients.State	DB	dbo.DimPatients.PatientState	DW	
Patients.dbo.Patients.ZipCode	DB	dbo.DimPatients.PatientZipCode	DW	
		dbo.DimPatients.StartDate	DW	
		dbo.DimPatients.PatientEndDate	DW	
		dbo.DimPatients.IsCurrent	DW	
		dbo.DimProcedures.ProcedureKey	DW	Surrogate Key
Patients.dbo.Procedures.ID	DB	dbo.DimProcedures.ProcedureID	DW	
Patients.dbo.Procedures.Name	DB	dbo.DimProcedures.ProcedureName	DW	
Patients.dbo.Procedures.Desc	DB	dbo.DimProcedures.ProcedureDesc	DW	
Patients.dbo.Procedures.Charge	DB	dbo.DimProcedures.ProcedureCharge	DW	
		dbo.DimClinics.ClinicKey	DW	Surrogate Key
DoctorSchedules.dbo.Clinics.ClinicID	DB	dbo.DimClinics.ClinicID	DW	
DoctorSchedules.dbo.Clinics.ClinicName	DB	dbo.DimClinics.ClinicName	DW	
DoctorSchedules.dbo.Clinics.City	DB	dbo.DimClinics.ClinicCity	DW	
DoctorSchedules.dbo.Clinics.State	DB	dbo.DimClinics.ClinicState	DW	
DoctorSchedules.dbo.Clinics.Zip	DB	dbo.DimClinics.ClinicZip	DW	
		dbo.DimShifts.ShiftKey	DW	Surrogate Key
DoctorSchedules.dbo.Shifts.ShiftID	DB	dbo.DimShifts.ShiftID	DW	
DoctorSchedules.dbo.Shifts.ShiftStart	DB	dbo.DimShifts.ShiftStart	DW	
DoctorSchedules.dbo.Shifts.ShiftEnd	DB	dbo.DimShifts.ShiftEnd	DW	
		dbo.DimDoctors.DoctorKey	DW	Surrogate Key
DoctorSchedules.dbo.Doctors.ShiftID	DB	dbo.DimDoctors.DoctorID	DW	
DoctorSchedules.dbo.Doctors.FirstName	DB	dbo.DimDoctors.DoctorFullName	DW	Concatenate first and last name
DoctorSchedules.dbo.Doctors.LastName	DB	dbo.DimDoctors.DoctorFullName	DW	
DoctorSchedules.dbo.Doctors.EmailAddress	DB	dbo.DimDoctors.DoctorEmailAddress	DW	
DoctorSchedules.dbo.Doctors.City	DB	dbo.DimDoctors.DoctorCity	DW	
DoctorSchedules.dbo.Doctors.State	DB	dbo.DimDoctors.DoctorState	DW	
DoctorSchedules.dbo.Doctors.Zip	DB	dbo.DimDoctors.DoctorZip	DW	
		dbo.DimDates.DateKey	DW	Surrogate Key
		dbo.DimDates.FullDate	DW	
		dbo.DimDates.MonthID	DW	
		dbo.DimDates.YearID	DW	
		dbo.DimDates.YearName	DW	
DoctorSchedules.dbo.DoctorShifts.DoctorSh	DB	dbo.FactDoctorShifts.DoctorsShiftID	DW	
dbo.DimDates.DateKey	DB	dbo.FactDoctorShifts.ShiftDateKey	DW	Lookup
dbo.DimClinics.ClinicKey	DB	dbo.FactDoctorShifts.ClinicKey	DW	Lookup
dbo.DimShifts.ShiftKey	DB	dbo.FactDoctorShifts.ShiftKey	DW	Lookup
dbo.DimDoctors.DoctorKey	DB	dbo.FactDoctorShifts.DoctorsKey	DW	Lookup
dbo.DimShifts.ShiftStart - dbo.DimShifts.ShiftE	DB	dbo.FactDoctorShifts.HoursWorked	DW	Delta of ShiftStart and ShiftEnd?
	DB	dbo.FactVisits.VisitKey	DW	Surrogate Key
Patients.dbo.Visits.Date	DB	dbo.FactVisits.DateKey	DW	
dbo.DimClinics.ClinicKey	DB	dbo.FactVisits.ClinicKey	DW	
dbo.DimPatients.PatientKey	DB	dbo.FactVisits.PatientKey	DW	
dbo.DimDoctors.DoctorKey	DB	dbo.FactVisits.DoctorKey	DW	
dbo.DimProcedures.ProcedureKey	DB	dbo.FactVisits.ProcedureKey	DW	
Patients.dbo.Visits.Charge	DB	dbo.FactVisits.ProcedureVisitCharge	DW	

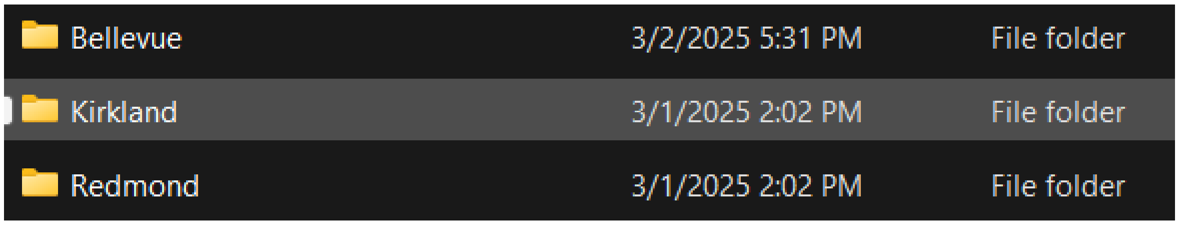
Files to Source Databases

Source Data	Source Type	Destination	Destination Typ	Transformations
20100102NewPatients.Fname	Excel	Patients.dbo.Patients.Fname	Database	NA
20100102NewPatients.Lname	Excel	Patients.dbo.Patients.Lname	Database	NA
20100102NewPatients.Email	Excel	Patients.dbo.Patients.Email	Database	Email address pattern validation
20100102NewPatients.Address	Excel	Patients.dbo.Patients.Address	Database	NA
20100102NewPatients.City	Excel	Patients.dbo.Patients.City	Database	NA
20100102NewPatients.State	Excel	Patients.dbo.Patients.State	Database	NA
20100102NewPatients.ZipCode	Excel	Patients.dbo.Patients.ZipCode	Database	NA
20100102Visits.Time	Excel	Patients.dbo.Visits.Date	Database	Convert from just time to Datetime
20100102Visits.Clinic	Excel	Patients.dbo.Visits.Clinic	Database	Bellevue file missing field, 1 must be added to rows in that file along with the field, will have to reorder column location in Redmond file

The Business' Daily Data

The business's daily data is the source data that is sent to the business's corporate office for them to manually process and upload to the database. The data is for Patients and Clinics that are both stored in separate CSV files in a folder of each city where the clinic is located. This is the original source data that begins the new ETL process.

Figure:

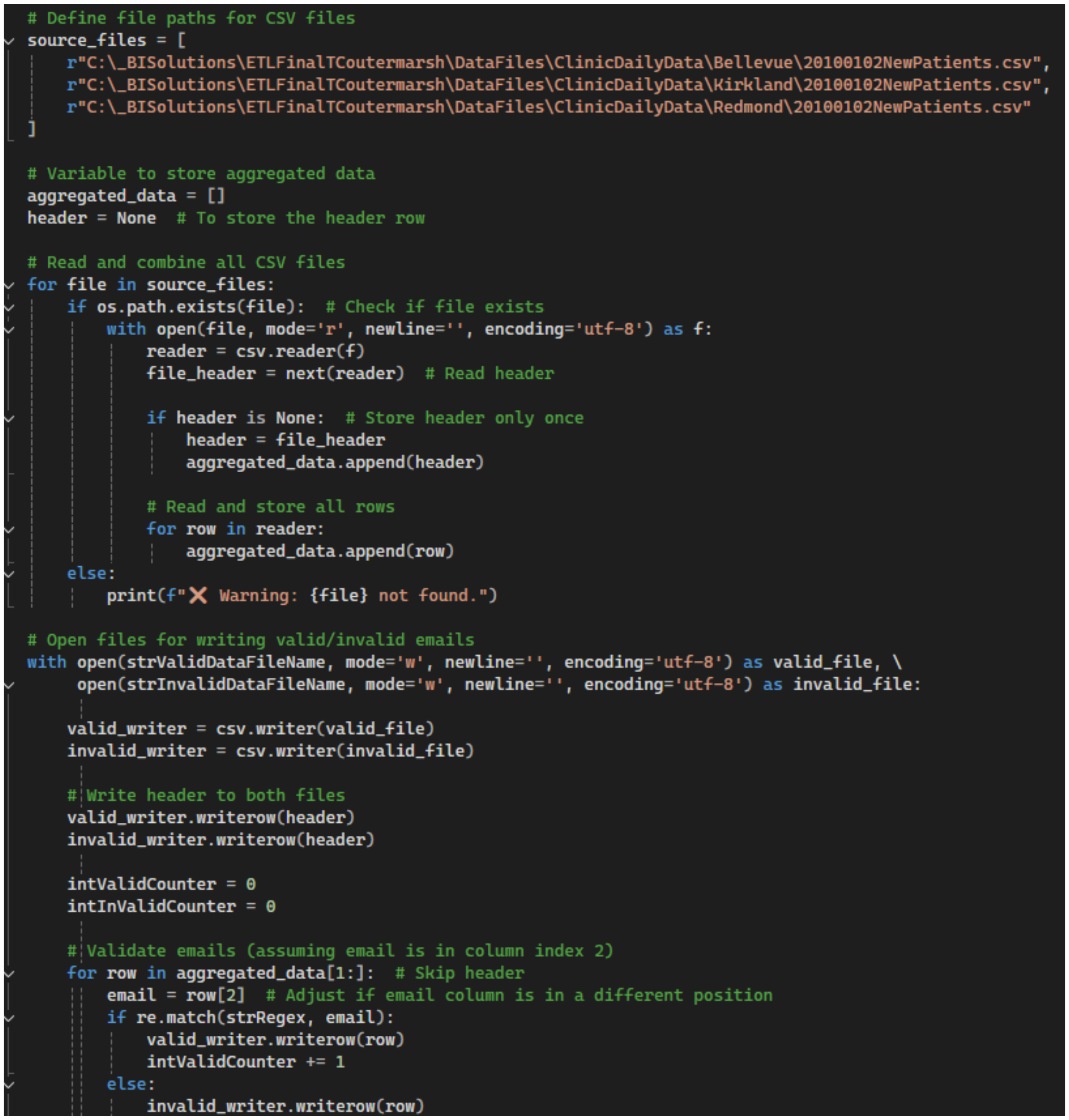


Bellevue	3/2/2025 5:31 PM	File folder
Kirkland	3/1/2025 2:02 PM	File folder
Redmond	3/1/2025 2:02 PM	File folder

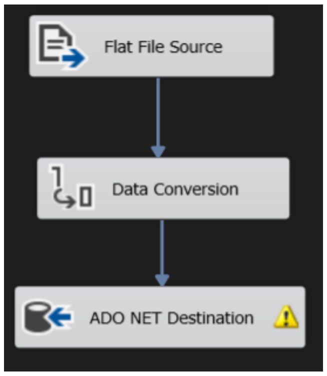
The File Based ETL

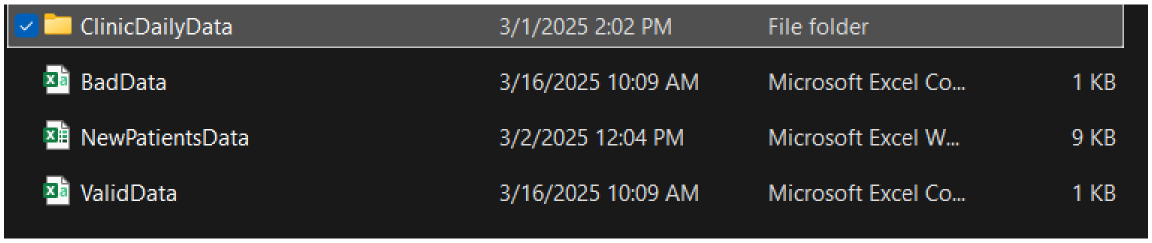
This component automates the current manual process of cleaning and uploading the business' daily data. The process consists of two scripts. A python script for processing the patient's data and a SQL script for processing the visits data. The python script defines the locations of the three patients CSV files. Within these files, there is an issue with the Email field that has inconsistent value entries. The python script combines all three CSV files into once, opens the files for writing and validates the data using a regular expression for emails. The regular expression compares the email values to the pattern of the regular expression for emails. If the email value matches the regular expression, it is placed in an excel file named ValidData. If it does not match, then it is placed in an excel file named BadData. After, a data flow task is used to connect to the destination database and the data in the ValidData excel file is inserted into the table in the database.

Figure:



```python
# Define file paths for CSV files
source_files = [
    r"C:\_BISolutions\ETLFinalTCoutermarsh\DataFiles\ClinicDailyData\Bellevue\20100102NewPatients.csv",
    r"C:\_BISolutions\ETLFinalTCoutermarsh\DataFiles\ClinicDailyData\Kirkland\20100102NewPatients.csv",
    r"C:\_BISolutions\ETLFinalTCoutermarsh\DataFiles\ClinicDailyData\Redmond\20100102NewPatients.csv"
]

# Variable to store aggregated data
aggregated_data = []
header = None  # To store the header row

# Read and combine all CSV files
for file in source_files:
    if os.path.exists(file):  # Check if file exists
        with open(file, mode='r', newline='', encoding='utf-8') as f:
            reader = csv.reader(f)
            file_header = next(reader)  # Read header

            if header is None:  # Store header only once
                header = file_header
                aggregated_data.append(header)

            # Read and store all rows
            for row in reader:
                aggregated_data.append(row)
    else:
        print(f"❌ Warning: {file} not found.")

# Open files for writing valid/invalid emails
with open(strValidDataFileName, mode='w', newline='', encoding='utf-8') as valid_file, \
     open(strInvalidDataFileName, mode='w', newline='', encoding='utf-8') as invalid_file:

    valid_writer = csv.writer(valid_file)
    invalid_writer = csv.writer(invalid_file)

    # Write header to both files
    valid_writer.writerow(header)
    invalid_writer.writerow(header)

    intValidCounter = 0
    intInValidCounter = 0

    # Validate emails (assuming email is in column index 2)
    for row in aggregated_data[1:]:  # Skip header
        email = row[2]  # Adjust if email column is in a different position
        if re.match(strRegex, email):
            valid_writer.writerow(row)
            intValidCounter += 1
        else:
            invalid_writer.writerow(row)
```

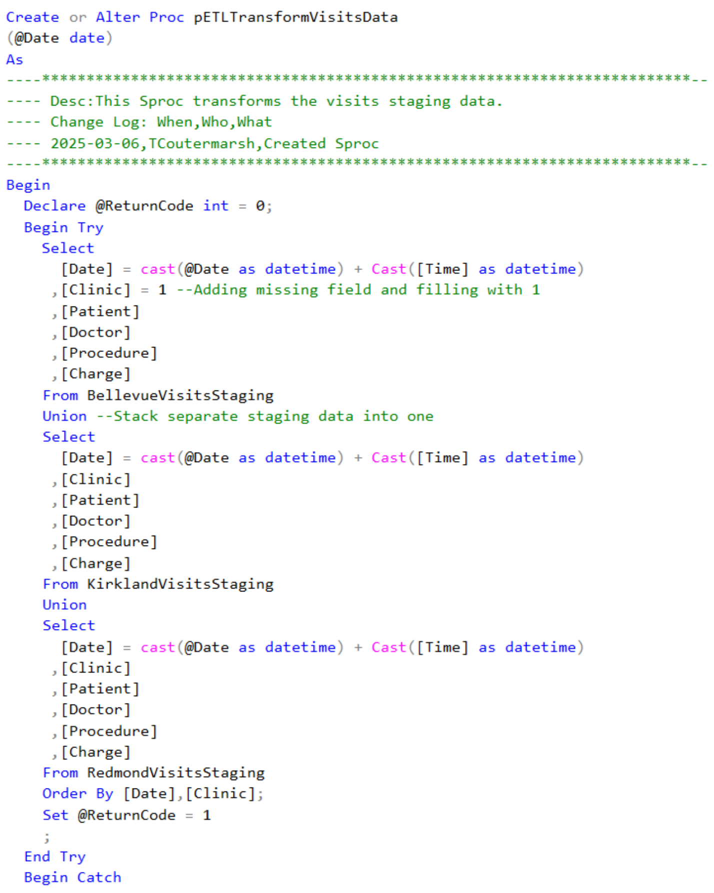




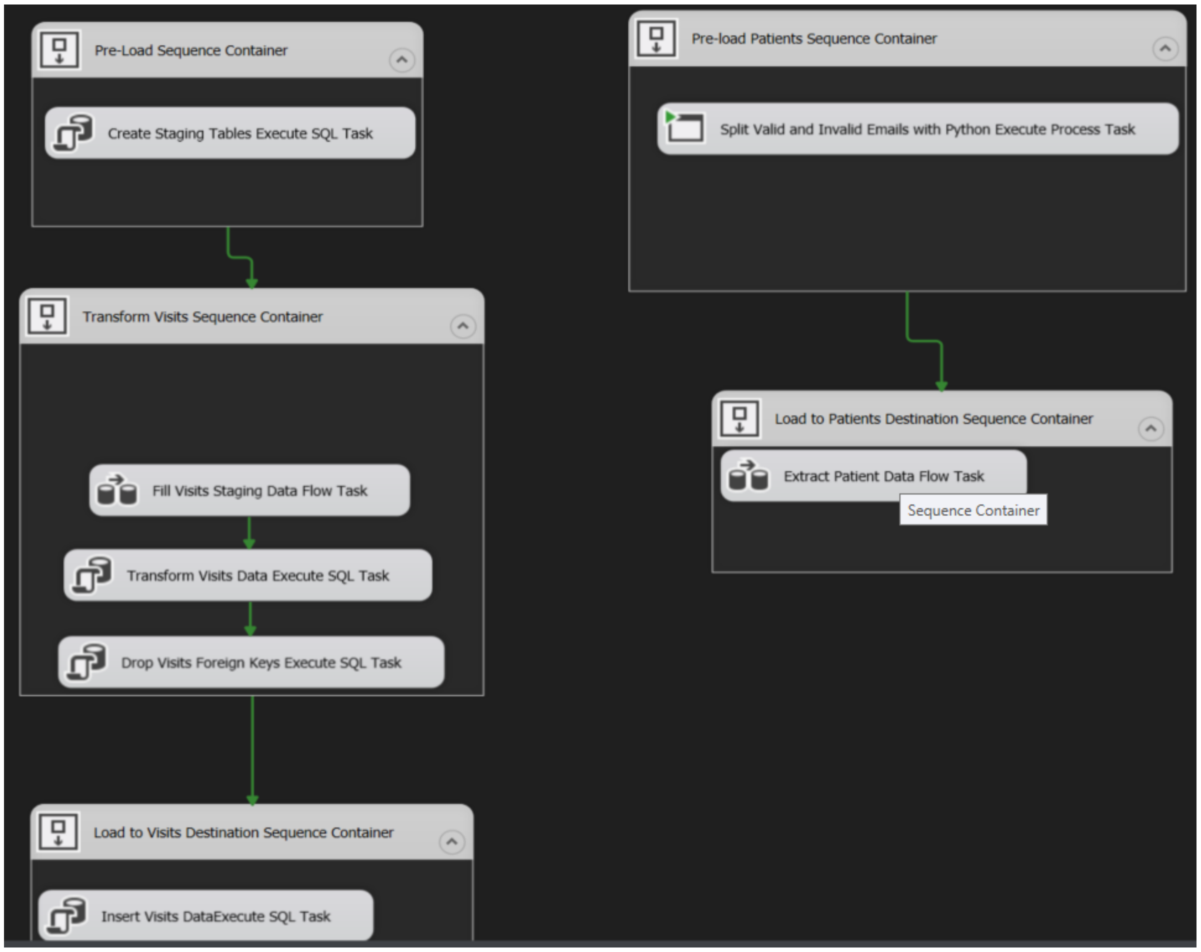
ClinicDailyData	3/1/2025 2:02 PM	File folder	
BadData	3/16/2025 10:09 AM	Microsoft Excel Co...	1 KB
NewPatientsData	3/2/2025 12:04 PM	Microsoft Excel W...	9 KB
ValidData	3/16/2025 10:09 AM	Microsoft Excel Co...	1 KB

The SQL script implements an ETL process for handling visits data from multiple clinics (Bellevue, Kirkland, Redmond). It begins by creating staging tables for temporary data storage, with Bellevue's table missing a Clinic field. The transformation procedure consolidates data from all staging tables, assigning a default value for the missing Clinic field in Bellevue and converting Time into a full DateTime value. Before inserting the transformed data into the Visits table, foreign key constraints are temporarily dropped to prevent conflicts. The data is then inserted using a transaction to maintain integrity, and the foreign keys are re-added afterward. Error handling is implemented throughout log issues and ensures a smooth ETL process. Within the same SSIS package, Execute SQL Tasks are used to execute each stored procedure within the script.

Firgure:



```
Create or Alter Proc pETLTransformVisitsData
(@Date date)
As
----*************************************************************************--
---- Desc:This Sproc transforms the visits staging data.
---- Change Log: When,Who,What
---- 2025-03-06,TCoutermarsh,Created Sproc
----*************************************************************************--
Begin
  Declare @ReturnCode int = 0;
  Begin Try
    Select
      [Date] = cast(@Date as datetime) + Cast([Time] as datetime)
     ,[Clinic] = 1 --Adding missing field and filling with 1
     ,[Patient]
     ,[Doctor]
     ,[Procedure]
     ,[Charge]
    From BellevueVisitsStaging
    Union --Stack separate staging data into one
    Select
      [Date] = cast(@Date as datetime) + Cast([Time] as datetime)
     ,[Clinic]
     ,[Patient]
     ,[Doctor]
     ,[Procedure]
     ,[Charge]
    From KirklandVisitsStaging
    Union
    Select
      [Date] = cast(@Date as datetime) + Cast([Time] as datetime)
     ,[Clinic]
     ,[Patient]
     ,[Doctor]
     ,[Procedure]
     ,[Charge]
    From RedmondVisitsStaging
    Order By [Date],[Clinic];
    Set @ReturnCode = 1
    ;
  End Try
  Begin Catch
```

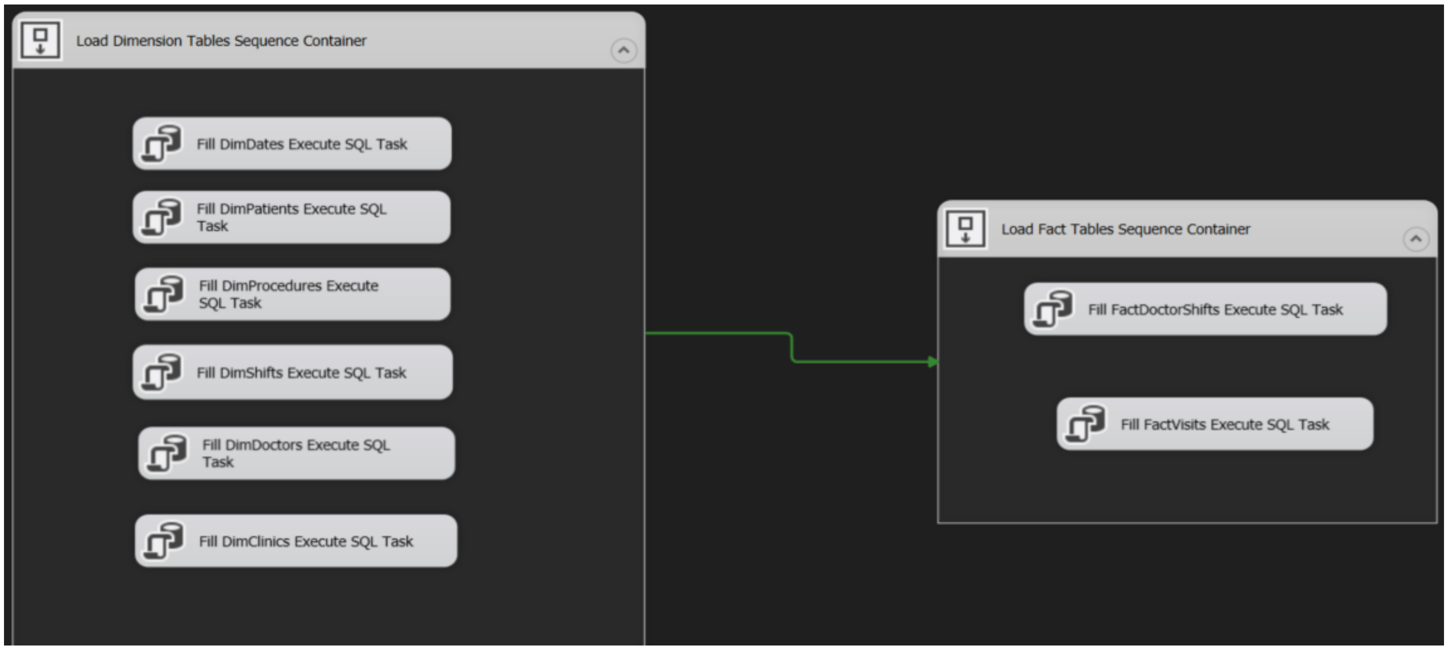

Pre-Load Sequence Container
Create Staging Tables Execute SQL Task
Transform Visits Sequence Container
Fill Visits Staging Data Flow Task
Transform Visits Data Execute SQL Task
Drop Visits Foreign Keys Execute SQL Task
Load to Visits Destination Sequence Container
Insert Visits DataExecute SQL Task
Pre-load Patients Sequence Container
Split Valid and Invalid Emails with Python Execute Process Task
Load to Patients Destination Sequence Container
Extract Patient Data Flow Task
Sequence Container

The Data Warehouse ETL

This part of the ETL process extracts data from the source database to transform and load the data into an OLAP formatted data warehouse using a SQL script. This SQL script sets up an **incremental ETL process** for a data warehouse, focusing on ETL logging, dimension table synchronization, and fact table updates. It begins by creating an ETLLog table, a view (vETLLog), and a stored procedure (pInsETLLog) to track ETL activities. The script then defines and synchronizes dimension tables: DimDates, DimPatients, DimProcedures, DimClinics, and DimShifts. Each dimension has an ETL view (vETLDim*) to extract and transform new or changed data from source tables and a stored procedure (pETLSyncDim*) to merge updates incrementally while handling inserts, updates, and deletions. The process ensures consistency using transactions and error handling. Additionally, it updates fact tables (FactEncounters and FactPayments), ensuring they reflect the latest transactional data. The ETL process logs successes and failures, maintaining a robust and efficient data pipeline. An SSIS package is also created to run each stored procedure from the script using Execute SQL Tasks.

Code sample:

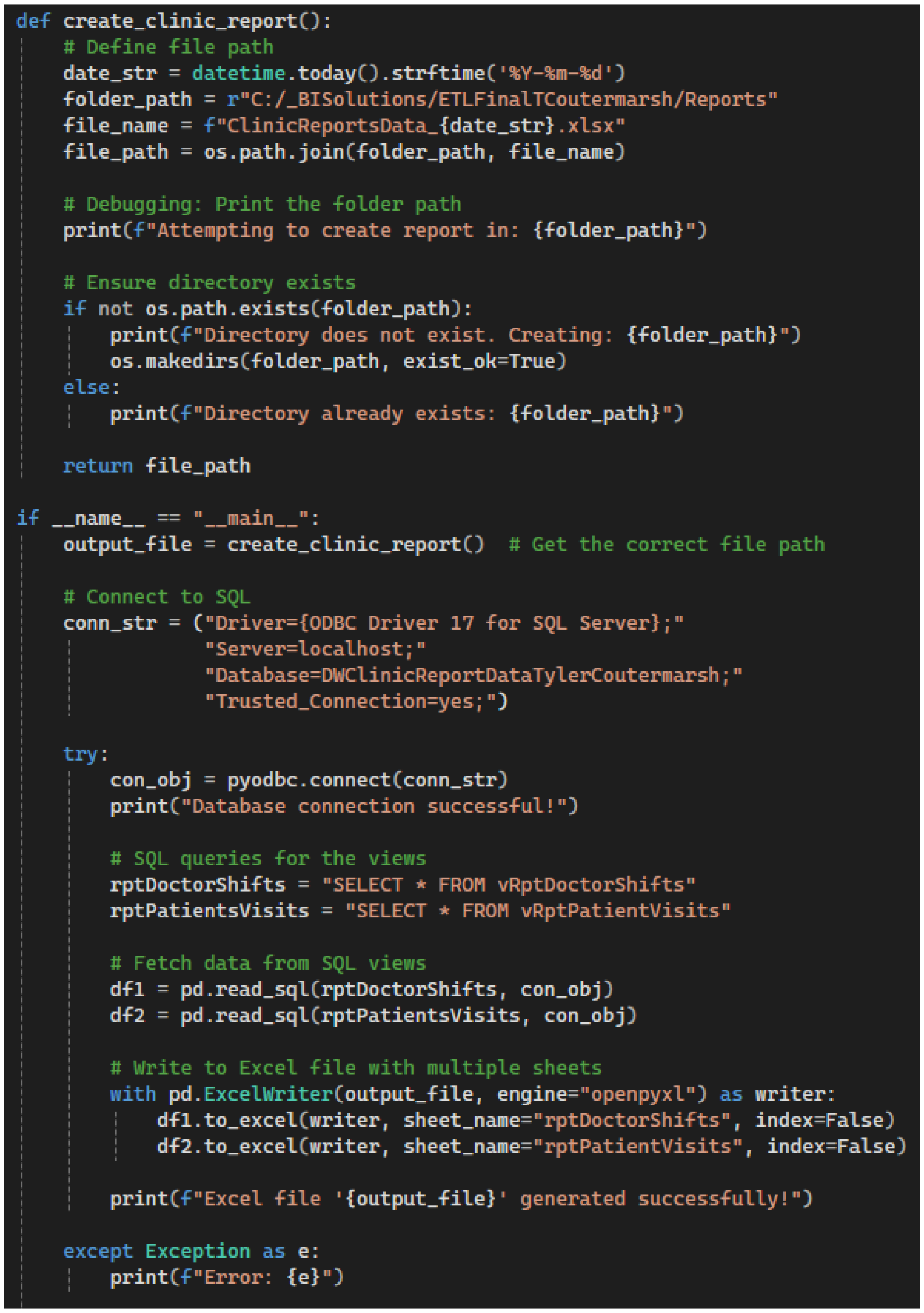
```sql
 If (OBJECT_ID('vRptPatientVisits') is not null) Drop View vRptPatientVisits;
go

 Create or Alter View vRptPatientVisits
/* Author: Tyler Coutermarsh
** Desc: Creates reporting view for patient visits
** Change Log: When,Who,What
** 2025-03-15, Tyler Coutermarsh, Created View.
*/
As
Select
 [VisitDate] = Cast(dd.FullDate as date)
,[PatientID] = dp.PatientID
,[PatientName] = dp.PatientFullName
,[DoctorID] = dds.DoctorID
,[DoctorName] = dds.DoctorFullName
,[ClinicKey] = dc.ClinicID
,[ClinicName] = dc.ClinicName
,[ClinicCity] = dc.ClinicCity
,[CliincState] = dc.ClinicState
,[ProcedureID] = dps.ProcedureID
,[ProcedureName] = dps.ProcedureName
,[ProcedureDesc] = dps.ProcedureDesc
,[ProcedureVisitsCharge] = fvs.ProcedureVistCharge
From FactVisits as fvs
Join DimDates as dd
 On fvs.DateKey = dd.DateKey
Join DimPatients as dp
 On fvs.PatientKey = dp.PatientKey
Join DimDoctors as dds
 On fvs.DoctorKey = dds.DoctorKey
Join DimClinics as dc
 On fvs.ClinicKey = dc.ClinicKey
Join DimProcedures as dps
 On fvs.ProcedureKey = dps.ProcedureKey;
 go
```


Load Dimension Tables Sequence Container
Fill DimDates Execute SQL Task
Fill DimPatients Execute SQL Task
Fill DimProcedures Execute SQL Task
Fill DimShifts Execute SQL Task
Fill DimDoctors Execute SQL Task
Fill DimClinics Execute SQL Task
Load Fact Tables Sequence Container
Fill FactDoctorShifts Execute SQL Task
Fill FactVisits Execute SQL Task

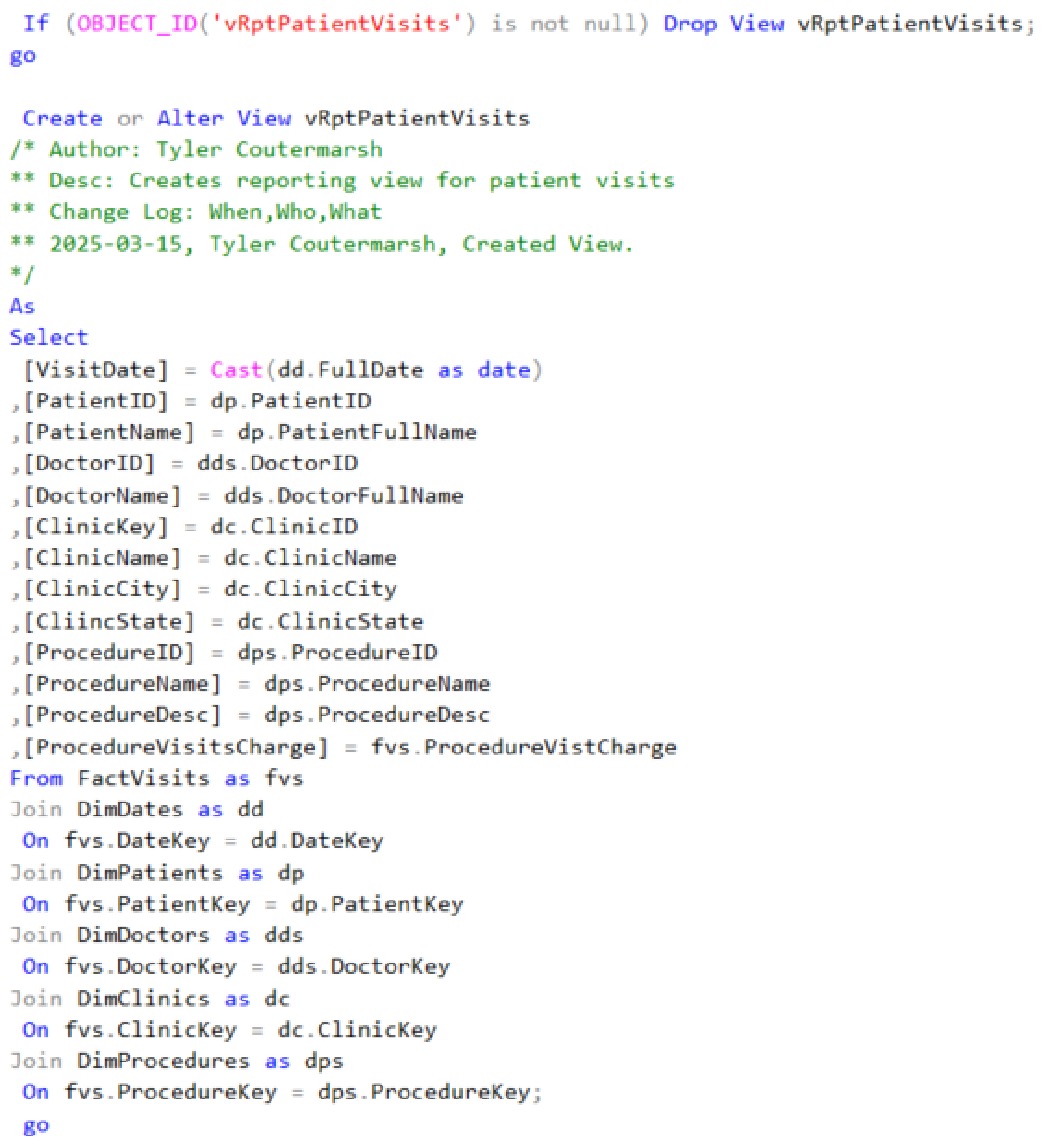
The Non-SQL ETL

This component creates reporting views from the new data warehouse that was created in the previous component. A python script is then used to connect to the views in the data warehouse and generates an excel report of the data from the views. This Python script implements an **incremental ETL process** for generating **clinic reporting views** by extracting data from a SQL Server database and exporting it to an Excel file. It starts by defining a function create_clinic_report() that determines the output file path based on the current date, ensuring the report is stored in a designated directory (C:/_BISolutions/ETLFinalTCoutermarsh/Reports). If the directory does not exist, it is created dynamically. The script then establishes a connection to the **DWClinicReportDataTylerCoutermarsh** database using pyodbc. It queries two reporting views, vRptDoctorShifts and vRptPatientVisits, retrieving data into Pandas DataFrames. The extracted data is written to an Excel file with separate sheets for each report using openpyxl. The process logs messages for debugging and error handling, ensuring smooth execution. Finally, the script safely closes the database connection after completion.

Code sample:



```python
def create_clinic_report():
    # Define file path
    date_str = datetime.today().strftime('%Y-%m-%d')
    folder_path = r"C:/_BISolutions/ETLFinalTCoutermarsh/Reports"
    file_name = f"ClinicReportsData_{date_str}.xlsx"
    file_path = os.path.join(folder_path, file_name)

    # Debugging: Print the folder path
    print(f"Attempting to create report in: {folder_path}")

    # Ensure directory exists
    if not os.path.exists(folder_path):
        print(f"Directory does not exist. Creating: {folder_path}")
        os.makedirs(folder_path, exist_ok=True)
    else:
        print(f"Directory already exists: {folder_path}")

    return file_path

if __name__ == "__main__":
    output_file = create_clinic_report()  # Get the correct file path

    # Connect to SQL
    conn_str = ("Driver={ODBC Driver 17 for SQL Server};"
                "Server=localhost;"
                "Database=DWClinicReportDataTylerCoutermarsh;"
                "Trusted_Connection=yes;")

    try:
        con_obj = pyodbc.connect(conn_str)
        print("Database connection successful!")

        # SQL queries for the views
        rptDoctorShifts = "SELECT * FROM vRptDoctorShifts"
        rptPatientsVisits = "SELECT * FROM vRptPatientVisits"

        # Fetch data from SQL views
        df1 = pd.read_sql(rptDoctorShifts, con_obj)
        df2 = pd.read_sql(rptPatientsVisits, con_obj)

        # Write to Excel file with multiple sheets
        with pd.ExcelWriter(output_file, engine="openpyxl") as writer:
            df1.to_excel(writer, sheet_name="rptDoctorShifts", index=False)
            df2.to_excel(writer, sheet_name="rptPatientVisits", index=False)

        print(f"Excel file '{output_file}' generated successfully!")

    except Exception as e:
        print(f"Error: {e}")
```

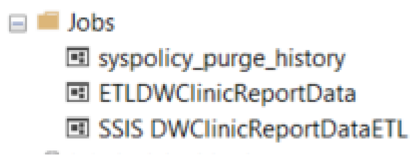


```sql
 If (OBJECT_ID('vRptPatientVisits') is not null) Drop View vRptPatientVisits;
go

 Create or Alter View vRptPatientVisits
/* Author: Tyler Coutermarsh
** Desc: Creates reporting view for patient visits
** Change Log: When,Who,What
** 2025-03-15, Tyler Coutermarsh, Created View.
*/
As
Select
 [VisitDate] = Cast(dd.FullDate as date)
,[PatientID] = dp.PatientID
,[PatientName] = dp.PatientFullName
,[DoctorID] = dds.DoctorID
,[DoctorName] = dds.DoctorFullName
,[ClinicKey] = dc.ClinicID
,[ClinicName] = dc.ClinicName
,[ClinicCity] = dc.ClinicCity
,[CliincState] = dc.ClinicState
,[ProcedureID] = dps.ProcedureID
,[ProcedureName] = dps.ProcedureName
,[ProcedureDesc] = dps.ProcedureDesc
,[ProcedureVisitsCharge] = fvs.ProcedureVistCharge
From FactVisits as fvs
Join DimDates as dd
 On fvs.DateKey = dd.DateKey
Join DimPatients as dp
 On fvs.PatientKey = dp.PatientKey
Join DimDoctors as dds
 On fvs.DoctorKey = dds.DoctorKey
Join DimClinics as dc
 On fvs.ClinicKey = dc.ClinicKey
Join DimProcedures as dps
 On fvs.ProcedureKey = dps.ProcedureKey;
 go
```

The SQL Server Agent Job

This component is used to automate the ETL process that incrementally loads data into the data warehouse. Using SQL Server Agent, a job is created to run stored procedures from the script daily at 1 am.

Figure:



Jobs
syspolicy_purge_history

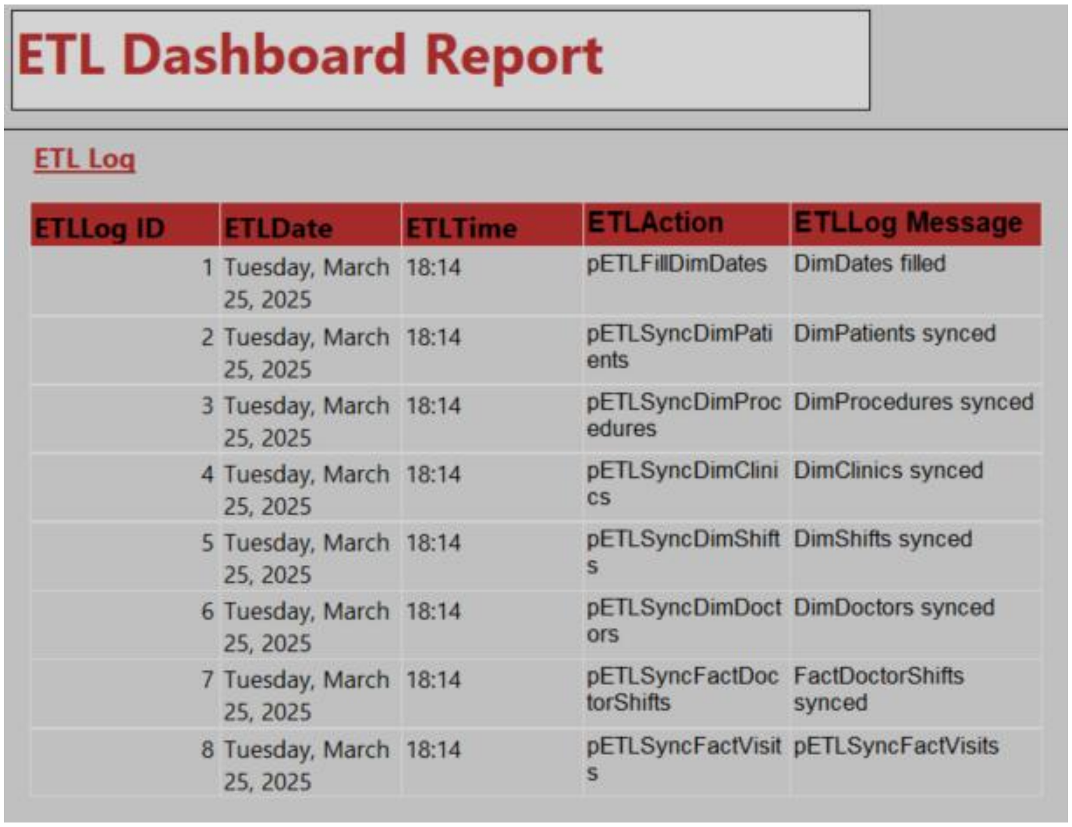
ETLDWClinicReportData

SSIS DWClinicReportDataETL

The ETL Reporting Dashboard

This component is a dashboard report that uses SQL Reporting Server that connects to the ETL logging views in the incremental ETL script for the data warehouse. The dashboard tracks the progress of the ETL process that is run daily through automation of the SQL Server Agent Job. The ETLLog Message shows if the stored procedure was completed successfully.

Figure:



ETL Dashboard Report

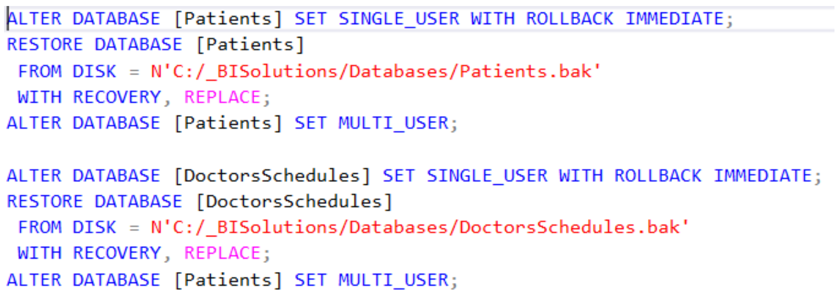
ETL Log

ETLLog ID	ETLDate	ETLTime	ETLAction	ETLLog Message
1	Tuesday, March 25, 2025	18:14	pETLFillDimDates	DimDates filled
2	Tuesday, March 25, 2025	18:14	pETLSyncDimPatients	DimPatients synced
3	Tuesday, March 25, 2025	18:14	pETLSyncDimProcedures	DimProcedures synced
4	Tuesday, March 25, 2025	18:14	pETLSyncDimClinics	DimClinics synced
5	Tuesday, March 25, 2025	18:14	pETLSyncDimShifts	DimShifts synced
6	Tuesday, March 25, 2025	18:14	pETLSyncDimDoctors	DimDoctors synced
7	Tuesday, March 25, 2025	18:14	pETLSyncFactDoctorShifts	FactDoctorShifts synced
8	Tuesday, March 25, 2025	18:14	pETLSyncFactVisits	pETLSyncFactVisits

The Database Restoration Script

This script is utilized if the source database needs to be restored on the machine the ETL process is being used on. This is the source database that the business' file data is being loaded into. The beginning of this ETL process.

Figure:



```
ALTER DATABASE [Patients] SET SINGLE_USER WITH ROLLBACK IMMEDIATE;
RESTORE DATABASE [Patients]
 FROM DISK = N'C:/_BISolutions/Databases/Patients.bak'
 WITH RECOVERY, REPLACE;
ALTER DATABASE [Patients] SET MULTI_USER;

ALTER DATABASE [DoctorsSchedules] SET SINGLE_USER WITH ROLLBACK IMMEDIATE;
RESTORE DATABASE [DoctorsSchedules]
 FROM DISK = N'C:/_BISolutions/Databases/DoctorsSchedules.bak'
 WITH RECOVERY, REPLACE;
ALTER DATABASE [Patients] SET MULTI_USER;
```